



# Web услуги. GET и POST заявки.

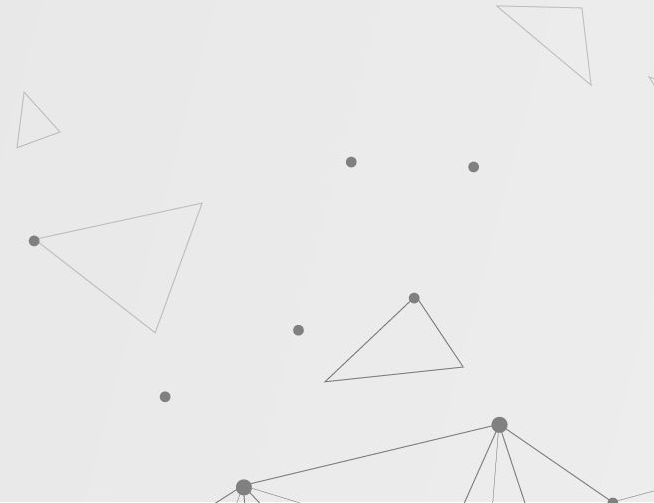
Иван Арабаджийски

Client-server архитектура.

HTTP Request Method.

HTTP Request.

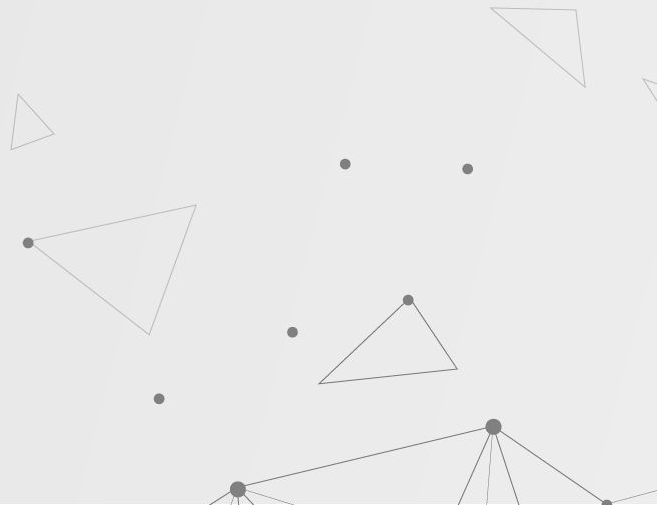
API



# REST услуги.

Софтуерен архитектурен стил, създаден от Roy Fielding през 2000 година. Всеки API, който следва REST шаблона за дизайн се нарича RESTful API.

По-просто казано REST API е лесен начин два компютъра да комуникират помежду си чрез HTTP по същия начин, по който комуникират клиентите и сървърите.



# Абревиатура?

- REpresentational
- State
- Transfer



# Какво е RESTful API?

Множество от архитектурни ограничения.

## Какво не е REST

Протокол или стандарт.



# Какво прави едно API RESTful?

- Клиент-сървър архитектура съставена от клиенти, сървъри и ресурси, които се достъпват през HTTP.
- Stateless клиент-сървър комуникация. Това означава, че информация не се запазва между отделните get заявки и всяка заявка е отделна и несвързана с другите.

Всеки ресурс се определя по следния начин:

- Основен URI, например `http://api.example.com/`
- Стандартен HTTP метод - `GET, POST, PUT, and DELETE`
- Тип на ресурса



# Добри практики



# Използваме JSON формат за изпращане и получаване на данни

JavaScript има вече готови начини методи за работа с JSON обекти (fetch API), защото те са основно направени за нея. Естествено всички други програмни езици като Python и PHP също имат библиотеки и методи за за JSON данни.

За да сме сигурни, че клиентите интерпретират JSON данните правилно трябва да сложим `Content-Type: application/json`

в рекуест хедъра

От страната на сървъра доста фреймърци вече правят това автоматично. Например `express.json() middleware`





# Използваме съществителни вместо глаголи в имената на endpoint-ите

Това е така, защото всеки ресурс се определя от път и метод. Методите вече са с имена глаголи - `GET`, `POST`, `PUT`, `PATCH`, and `DELETE`

Това са основните CRUD (create, read, update, delete) операции.

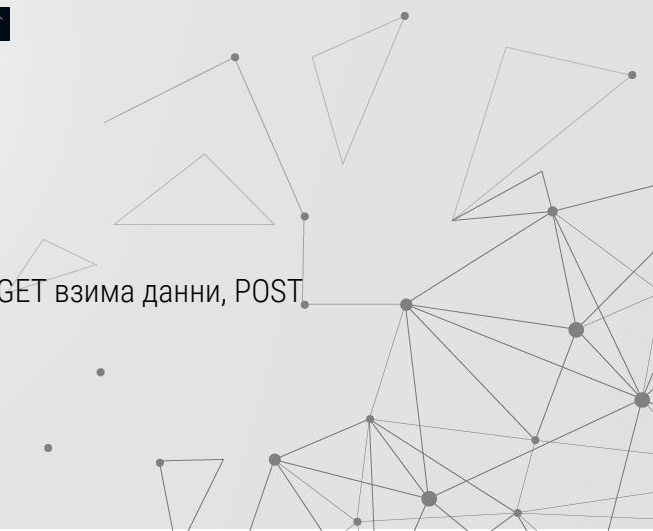
Пример: един endpoint не трябва да изглежда така

```
`https://mysite.com/getPosts` or `https://mysite.com/createPost`
```

А по-скоро така:

```
`https://mysite.com/posts`
```

Накратко, трябва да оставим HTTP методите да описват какво прави endpoint-а. GET взима данни, POST създава данни, PUT променя, DELETE изтрива.



# Имената на колекциите да са в множествено число

```
https://mysite.com/post/123 -> https://mysite.com/posts/123
```



# Правилни статус кодове в обработка на грешки

STATUS CODE RANGE	MEANING
100-199	Informational Responses For example, 102 indicates the resource is being processed
300-399	Redirects For example, 301 means Moved permanently
400-499	Client-side errors 400 means bad request and 404 means resource not found
500-599	Server-side errors For example, 500 means an internal server error

# Използване на вложени пътища за да покажем релация

`https://mysite.com/posts/author`` - Би било валидно влагане.

`https://mysite.com/posts/postId/comments`` Също

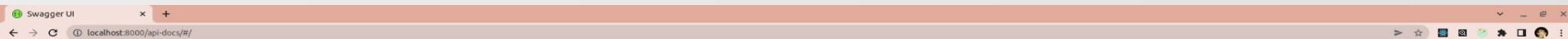


# Използване на филтриране, сортиране и номериране на страници за да вземем необходимите данни

```
https://mysite.com/posts?tags=javascript`
```



# Добра документация



## Example Express API with Swagger 0.1.0 OAS3

This is a simple CRUD API application made with Express and documented with Swagger

[Example - Website](#)  
[Send email to Example](#)  
[MIT](#)

Servers

### Books

- GET** `/books/` Lists all the books
- POST** `/books/` Creates a new book
- GET** `/books/{id}` Gets a book by id
- PUT** `/books/{id}` Updates a book
- DELETE** `/books/{id}` Deletes a book by id

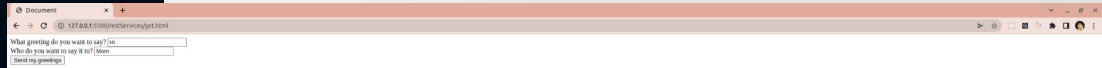
Schemas

# Изпращане на данни чрез HTML форми



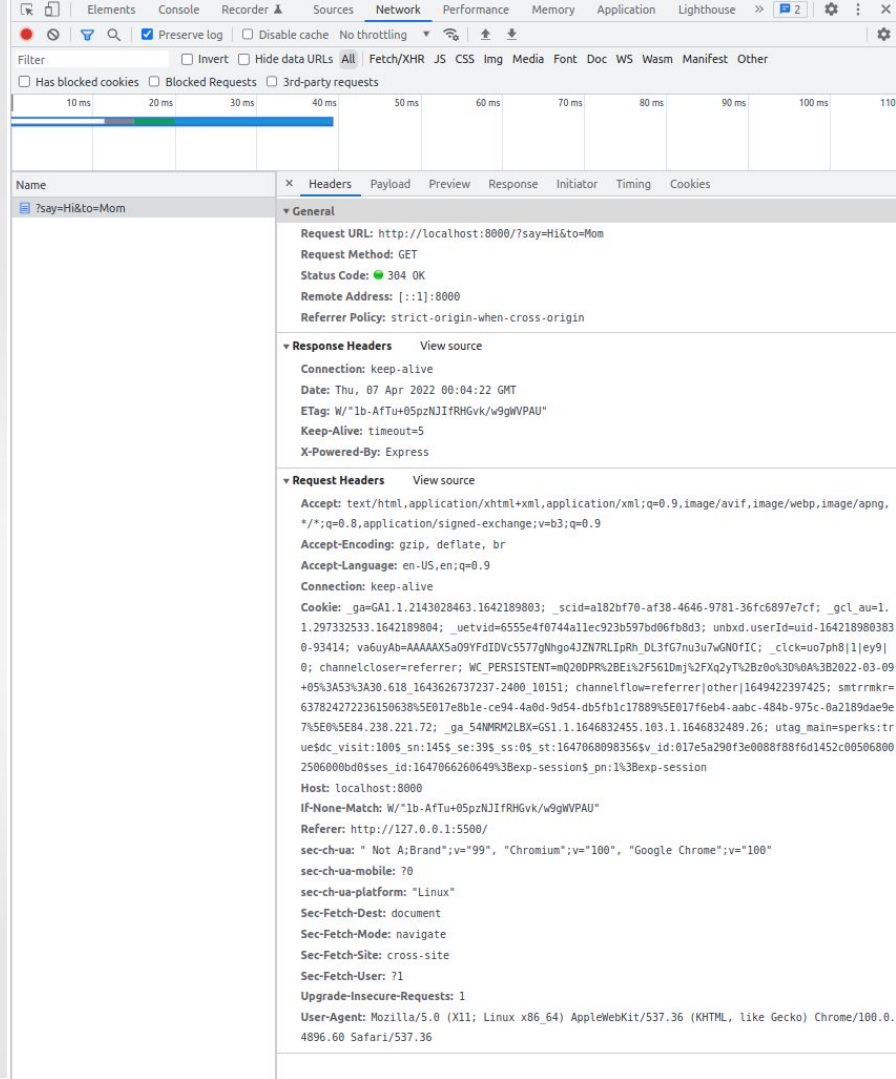
# GET

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Document</title>
8   </head>
9   <body>
10    <form action="http://localhost:8000/" method="GET">
11      <div>
12        <label for="say">What greeting do you want to say?</label>
13        <input name="say" id="say" value="Hi" />
14      </div>
15      <div>
16        <label for="to">Who do you want to say it to?</label>
17        <input name="to" id="to" value="Mom" />
18      </div>
19      <div>
20        <button>Send my greetings</button>
21      </div>
22    </form>
23  </body>
24 </html>
```





# А така изглежда когато се изпрати заявка



The screenshot shows the Chrome DevTools Network tab. A request to `http://localhost:8000/?say=Hi&to=Mom` is selected. The request is a GET with a 304 OK status. The response headers include `Connection: keep-alive`, `Date: Thu, 07 Apr 2022 00:04:22 GMT`, and `ETag: W/"1b-AFTu+05pzNJIfrHGvk/w9gWVPAU"`. The request headers include `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9`, `Accept-Encoding: gzip, deflate, br`, and `Accept-Language: en-US,en;q=0.9`. The cookie is `ga=GA1.1.2143028463.1642189803; _scid=a182bf70-af38-4646-9781-36fc6897e7cf; gcl_au=1.1.297332533.1642189804; _uetvid=6555e4f0744a11ec923b597bd06fb8d3; unbxid.userid=uid-1642189803830-93414; va6uyAb=AAAAAX5a09YFfIDVc5577gNhgo4JZNR7LlPpRh_DL3fG7nu3u7MGNOfIC; _clck=uo7ph8|1|ey9|0; channelcloser=referrer; WC_PERSISTENT=mQ200PRn%2BEi%2F5610m%2FXq2y7%2Bz0%3D%0A%3B2022-03-09+05%3A53%3A30.618.1643626737237-2400.10151; channelflow=referrer|other|1649422397425; smtrmk=637824272236150638%5E017e8b1e-ce94-4a0d-9d54-db5fb1c17889%5E017f6eb4-aabc-484b-975c-0a2189dae9e7%5E0%5E84.238.221.72; ga_54NMRM2LBX=GS1.1.1646832455.103.1.1646832489.26; utag_main=sperks:tr ue$dc_visit:1005_sn:145$_se:39$_ss:0$_st:1647068098356$sv_id:017e5a290f3e008f88f6d1452c00506800250600bd05ses_id:1647066260649%3Bexp-session$pn:1%3Bexp-session`. The host is `localhost:8000` and the referer is `http://127.0.0.1:5500/`. The user agent is `Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.4896.60 Safari/537.36`.

# POST заявката изпраща и тяло

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Document</title>
8   </head>
9   <body>
10    <form action="http://localhost:8000/" method="POST">
11      <div>
12        <label for="say">What greeting do you want to say?</label>
13        <input name="say" id="say" value="Hi" />
14      </div>
15      <div>
16        <label for="to">Who do you want to say it to?</label>
17        <input name="to" id="to" value="Mom" />
18      </div>
19      <div>
20        <button>Send my greetings</button>
21      </div>
22    </form>
23  </body>
24 </html>
```

The screenshot shows the Chrome DevTools Network tab. The 'Network' tab is selected, and a request to 'localhost' is highlighted. The 'Payload' tab is active, showing the form data: 'say: Hi' and 'to: Mom'. The 'Headers' tab is also visible, showing the request method as 'POST'.

Name
localhost

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
localhost		<b>Form Data</b> say: Hi to: Mom	view source	view URL-encoded			

## И как създавам REST API?

# Expressjs

```
iarabadzhiyski@Astea-20158:~$ node -v  
v14.18.0  
iarabadzhiyski@Astea-20158:~$ npm -v  
6.14.15
```



```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm init -y
Wrote to /home/iarabadzhiyski/js/lectures/restServices/expressExample/package.json:
```

```
{
  "name": "expressexample",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm install express
```

```
added 50 packages, and audited 51 packages in 2s
```

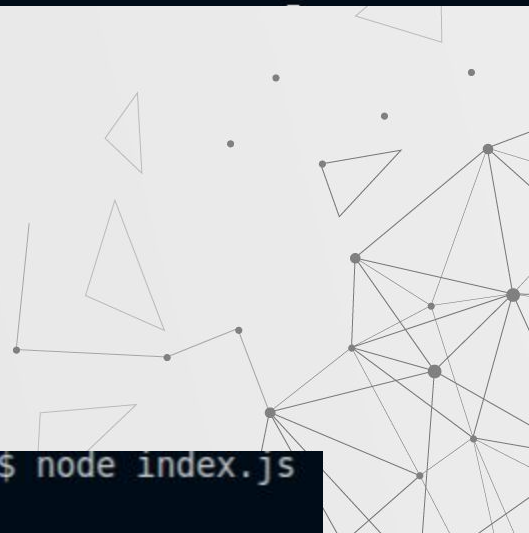
```
2 packages are looking for funding
  run `npm fund` for details
```

```
found 0 vulnerabilities
```

```
1  const express = require("express");
2
3  const app = express();
4  const port = 3000;
5
6  app.get("/", (req, res) => {
7    res.send("Express + TypeScript Server");
8  });
9
10 app.listen(port, () => {
11   console.log(`[server]: Server is running at https://localhost:${port}`);
12 });
13
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ node index.js
```

```
[server]: Server is running at https://localhost:3000
```



Express + TypeScript Server

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm i -D typescript @types/express @types/node
```

```
added 10 packages, and audited 61 packages in 3s
```

```
2 packages are looking for funding  
run `npm fund` for details
```

```
found 0 vulnerabilities
```

```
"devDependencies": {  
  "@types/express": "^4.17.13",  
  "@types/node": "^17.0.23",  
  "typescript": "^4.6.3"  
}
```

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npx tsc --init
```

```
Created a new tsconfig.json with:
```

```
target: es2016  
module: commonjs  
strict: true  
esModuleInterop: true  
skipLibCheck: true  
forceConsistentCasingInFileNames: true
```

```
You can learn more at https://aka.ms/tsconfig.json
```

TS

```
import express, { Express, Request, Response } from "express";

const app: Express = express();
const port = process.env.PORT;

app.get("/", (req: Request, res: Response) => {
  res.send("Express + TypeScript Server");
});

app.listen(port, () => {
  console.log(`[server]: Server is running at https://localhost:\${port}`);
});
```



You can learn more at <https://aka.ms/tscconfig.json>

```
iarabadzhiyski@Astea-20158:~/js/lectures/restServices/expressExample$ npm install -D concurrently nodemon
```

```
added 131 packages, and audited 192 packages in 10s
```

```
20 packages are looking for funding  
run `npm fund` for details
```

```
found 0 vulnerabilities
```

```
"scripts": {  
  "build": "npx tsc",  
  "start": "node dist/index.js",  
  "dev": "concurrently \"npx tsc --watch\" \"nodemon -q dist/index.js\"",  
},
```





# Какво друго можем да правим с express?

```
app.get("/", (req: Request, res: Response) => {
  console.log(req.query);
  res.send("Express + TypeScript Server");
});

app.post("/", (req: Request, res: Response) => {
  console.log(req.body);
  res.sendFile(path.join(__dirname, "htmlTemplates/index.html"));
});

app.post("/upload", upload.single("myFile"), (req: Request, res: Response) => {
  if (!req.file) {
    return res.status(401).json({ error: "Please provide an image" });
  }
  const imagePath = path.join(__dirname, `public/${req.file.originalname}`);
  createWriteStream(imagePath).write(req.file.buffer);
  res.redirect(`/media/${req.file.originalname}`);
});
```

# What the middleware?

```
app.use(urlencoded({ extended: true }));
app.use(express.json());

function logMethod(req: Request, res: Response, next: NextFunction) {
  console.log("Request Type:", req.method);
  next();
}

app.use((req, res, next) => {
  console.log("Time:", Date.now());
  next();
});

app.use(logMethod);
```

# Автентикация?

```
function authenticateJWT(req: Request, res: Response, next: NextFunction) {
  const authHeader = req.headers.authorization;
  if (authHeader && authHeader !== "null") {
    const token = authHeader.split(" ")[1];
    jwt.verify(token, JWT_KEY, (err: any, user: any) => {
      if (err) {
        return res
          .status(403)
          .send({ success: false, message: "Token Expired" });
      }
      // req.user = user;
      next();
    });
  } else {
    res.status(403).json({ success: false, message: "Unauthorized" });
  }
}
```

# Ние искаме RESTful API -> трябва да връщаме JSON.

```
app.get("/", (req: Request, res: Response) => {
  console.log(req.query);
  res.json({ name: "Express + TypeScript Server", ...req.query });
});

app.post("/", (req: Request, res: Response) => {
  console.log(req.body);
  res.json({ ...req.body });
});

app.post("/upload", upload.single("myFile"), (req: Request, res: Response) => {
  if (!req.file) {
    return res.status(401).json({ error: "Please provide an image" });
  }
  const imagePath = path.join(__dirname, `public/${req.file.originalname}`);
  createWriteStream(imagePath).write(req.file.buffer);
  res.json({ path: `http://localhost:8000/media/${req.file.originalname}` });
});
```

# Как обикновено изглежда един RESTful API?

```
type Book = {
  id?: string;
  createdAt?: number;
  title: string;
  author: string;
  finished: boolean;
};

const harryPotterAndThePhilosophersStone: Book = {
  id: "0",
  title: "Harry Potter and the Philosopher's Stone",
  author: "J. K. Rowling",
  finished: true,
};

const books = [harryPotterAndThePhilosophersStone];

app.get("/books/", (req: Request, res: Response) => {
  res.json({ books });
});

app.get("/books/:id", (req: Request, res: Response) => {
  const book = books.find((book) => book.id === req.params.id);
  if (!book) {
    return res.status(404).send();
  }
  res.json({ book });
});
```

```
function createNewBook({
  title,
  author,
  finished,
}): Pick<Book, "title" | "author" | "finished">: Book {
  // automatically create an id, use uuid or let the db generate one
  return {
    id: Math.random().toString(),
    title,
    author,
    finished,
    createdAt: Date.now(),
  };
}

function saveItToTheDatabase(book: Book) {
  return true;
}

app.post("/books", (req: Request, res: Response) => {
  // create a new book and save it to the database
  const book = createNewBook(req.body);
  if (!book) {
    return res.status(400).send();
  }
  const saved = saveItToTheDatabase(book);
  if (!saved) {
    return res.status(400).send();
  }
  res.json({ book });
});
```

# Swagger?



# Въпроси?

